

PI Manager Summary (UM)

[Plugin Manager Github Wiki](#) Extensive documentation

Design Goals

1. Make plugins easily discoverable for users.
2. Support simple, one-button click plugin installation
3. Not use elevated privileges (partly implied by above)
4. Obsolete the need to package plugins.
5. Be compatible with existing plugin installations
6. Improve build and deploy workflow for plugins.

Design Goals - Additional & Later

1. Simple, one-button click install for user.
2. Simple, one-button click uninstall for user.
3. Common UX on all platforms.
4. All devs: Simple, well-defined model of what a plugin is installation-wise.
5. Ocpn devs: Platform-independent code (mostly).
6. Plugins devs: Make it simpler to provide plugins.

Alec's Discussion

The simple, platform-independent one-click user actions are simply not feasible using native packages. Reasons include that uninstall just isn't possible, technical problems when communicating with privileged code, different or missing GUIs and fragmented Linux solutions.

For plugin developers the new model basically means that they need to support make install instead of providing packages. This is much easier, and cmake gives tons of support.

For opencpn we get a common, unified platform with not that much platform-dependent code (estimated < 300 lines).

Both ocpn and plugin devs benefits from the more restricted model of what a plugin actually is installation-wise. When using platform packages a plugin could basically install all sorts of files in all sorts of places, run scripts on install/uninstall, pull in platform dependencies etc. Even if most (almost all?) plugins only installs a library and some data files, any design using platform packages will be hampered by what a platform package might do.

All this boils down to a solution similar to the browsers, IDE:s, multimedia tools etc., out there: use an application-specific format for the plugin (.tar.gz in this case).

Dave's Summary

Operation:

1. There is a metadata file called "ocpn-plugins.xml" which contains a description, tarball URLs, and potentially other stuff for a set of plugins.
2. The ocpn-plugins.xml file is created by merging the files in plugins/metadata. Each of these files represents a plugin build for a given platform.
3. When placed in the OCPN private data directory, OCPN will (at startup) notice the metadata, and offer a GUI interface to "manage" the so-described plugins.
4. Each plugin metadata section in ocpn-plugins.xml may (will?) be platform specific.
5. Legacy (user-managed) plugin installers (by .exe, .dmg, or PPA) can coexist, but are in no way "managed" by the new GUI.

Control issues:

1. I assume OCPN can download at will a new updated copy of ocpn-plugins.xml, on user request.
2. Today, the act of "publishing" a new approved plugin for OCPN consists of creating an entry on the opencpn.org website describing the plugin, and providing links to (soon to be called) legacy installers for user integration and management.
3. In the new scheme, "publishing" a plugin includes all of (6) above, as well as creating and promulgating a new release of ocpn-plugins.xml, properly versioned in some unclear way.

Hosting questions:

1. We collectively wonder where to host the ocpn-plugins.xml files, with some kind of version control, access privileges, etc. NOTE: The repository for ocpn-plugins.xml is <https://github.com/OpenCPN/plugins>
2. What are the policies as regards this critical file? Who can update the official copy, etc? Dave and several other developers have admin access to this repository.
3. We also wonder where, or if, to assertively host the tarballs, and other blobs required for installation, for each plugin.

Ideas:

1. Start a new project on github called OpenCPN/plugins, with the same maintainers as OpenCPN/OpenCPN.
2. Move the parts related to building ocpn-plugins.xml to this new project. This includes plugins/metadata, plugins/icons, parts of tools/and parts of CMakeLists.txt which creates ocpn-plugins.xml
3. The new project basically "exports" ocpn-plugins.xml. It has an independent versioning scheme, perhaps as simple as a single number.

4. Each release of OpenCPN is distributed with a fixed version of ocpn-plugins.xml.
5. The code in OpenCPN [PR #1457](#) is updated so it can update ocpn-plugins.xml by downloading it from from github on user request.

Alec's Initial Summary

This is about the new Plugin Installer as proposed in [\[1\]](#)

1. For each plugin, the installer needs an XML metadata file and a tarball containing what to install. Together with existing packages (.exe, .img, .deb etc.) this makes three files per plugin.
2. I have forked four example plugins to make them build the necessary stuff for the installer: oesenc [\[4\]](#) , squiddio [\[3\]](#) , radar [\[2\]](#) and bsb4 [\[5\]](#).
3. I have made two pull requests based on my forks: squiddio (merged) and radar_pi [\[2\]](#).
4. The question now discussed is basically how to deploy the stuff built in various ci builders. This is raised as an issue in [\[3\]](#).
5. If this should be handled on a global level (I actually don't see the need at this point) the discussion must IMHO be moved to the overall opencpn project.'

Cheers! Alec

[1] PR #1457 <https://github.com/OpenCPN/OpenCPN/pull/1457>

[2] Radar_pi https://github.com/opencpn-radar-pi/radar_pi/pull/120

[3] Squiddio_pi https://github.com/mauroc/squiddio_pi/issues/71

[4] Oesenc_pi https://github.com/leamas/oesenc_pi/

[5] Bsb4_pi https://github.com/leamas/bsb4_pi/

Questions

Explain the advantages of cloudsmith ci?

The basic counter-question is: compared to bintray, github or what? I can see some general advantages:

1. It actually seems mature.
2. We can create multiple repos for installer stuff (metadata + tarballs), test builds and legacy manual installers. Seems hard on github.
3. With a separate repo for manual packages it becomes more user-friendly (4 files /release instead of 12 for a typical plugin).
4. Cloudsmith has a rich administrative interface.
 1. For example, the retention policy could be set per repo to keep all, keep the latest etc.
 2. This is important to keep the list of test builds manageable.
5. Thanks to a clever scheme there is no need for encrypted deployment keys. Makes life easier for plugin devs.

Still, there is no project decision to use cloudsmith, it's a per-plugin decision to make or not. The only condition is that the plugin artifacts are downloadable from an url, on cloudsmith or elsewhere.

Pros Cloudsmith

1. Mature
2. Free to opensource
3. No need for encrypted deployment keys, which makes life easier for plugin devs.
4. Integrates well with github and circleci.
5. Rich administrative interface.
 1. Retention policy could be set per repo to keep all, keep the latest etc.
 2. Important to keep the list of test builds manageable.
 3. Can make an organization account and can manage & share permissions (and repositories) for that account.
 1. Opencpn organization on github which creates the Cloudsmith Opencpn Organization?
 2. Organization account url <https://cloudsmith.io/~opencpn/repos/plugins/packages/>
 3. Good for users, easier publishing to Opencpn.org Downloads page links.
 4. Since free/open-source, no billing is required.
 5. Decide who "owns" the opencpn org on Cloudsmith.
 6. Owners can then invite the rest of the plugin developers to the org on Cloudsmith
 1. give them specific read (or write) access to specific repositories.
 2. Maybe each plugin has its own repository, or you create a centralized repository for all of the plugins.
 3. That would be up to the opencpn developers.
 7. Note: Alec's PI Installer does not require centralized repository, flexible source.
6. Allows creation of multiple repositories to host the stable, unstable, manual versions of the binary and xml files.
7. With a separate repo for manual packages it becomes more user-friendly (4 files /release instead of 12 for a typical plugin).
8. Good search features
9. GitHub Actions (new service available to opensource) should be able to export to cloudsmith

Cons Cloudsmith

1. Not a fully integrated github service.
2. Requires another setup and management.

Pros GitHub

1. Integrates fully with GitHub
2. Releases are linked back to Commit Hash
3. Release are serially presented, but searchable

Cons GitHub

1. Release Tab Repository is serially presented.
2. Seems hard to create multiple repos on github.
3. Cannot create multiple repositories in a single account. (stable, unstable, manual)

We collectively wonder where to host the `ocpn-plugins.xml` files

With some kind of version control, access privileges, etc. What are the policies as regards this critical file? Who can update the official copy, etc? Some ideas:

1. Start a new project on github called **OpenCPN/plugins**, with the same maintainers as **OpenCPN/OpenCPN**.
2. Move the parts related to building **ocpn-plugins.xml** to this new project. This includes
 1. plugins/metadata
 2. plugins/icons
 3. parts of tools
 4. parts of CMakeLists.txt which creates ocpn-plugins.xml
3. The new project basically “exports” ocpn-plugins.xml. It has an independent versioning scheme, perhaps as simple as a single number.
4. Each release of OpenCPN is distributed with a fixed version of ocpn-plugins.xml.
5. The code in #1457 is updated so it can *update ocpn-plugins.xml* by downloading it from from github *on user request*.

Why is CircleCI appropriate to use?

1. The killer feature is the ability to log in to failed builds when running into trouble.
2. Tokens and encryption is easier, particularly for Win Dev (The simplified token/no encryption is actually cloudsmith).
3. Circleci seems quite fast, although Travis is good too.
4. Circleci information menus are very good, example https://circleci.com/gh/rgleason/squiddio_pi/140

From:
<https://opencpn.org/wiki/dokuwiki/> - **OpenCPN Manuals**

Permanent link:
https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:developer_manual:pi_installer_summary

Last update: **2021/08/18 02:37**

